194-00316TPW

Data Compression Study for the ECS Project

White Paper
Working Paper Version 1

Working paper - Not intended for formal review or Government approval.

June 94

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Howard Feil /s/ 8/5/94

Howard Feil Date
EOSDIS Core System Project

SUBMITTED BY

Stephen Fox /s/ 8/1/94

Steve Fox, SDPS Office Manager Date
EOSDIS Core System Project

Hughes Applied Information Systems
Landover, Maryland

This page intentionally left blank.

Contents

This page intentionally left blank.
(Use this when the Contents ends on an odd page.)

1. Introduction

1.1 Purpose

During the EOSDIS era, there will be significant volumes of data generated, transmitted, and managed on a daily basis. Data compression is one tool that can be used to overcome the problems encountered with data transmission, storage, and dissemination of such large volumes of data. This paper will give an overview of the current (Version 0) use of data compression at the DAAC sites and give a justification for the current methods of data compression. The paper will continue with an explanation of current data compression techniques in industry, and the paper will begin to give recommendations and justifications for possible data compression techniques for the ECS data sets. The paper concludes with recommendations for future studies.

1.2 Organization

This paper is organized as follows:

- Section 1 presents the purpose of the document and its organization.
- Section 2 gives an overview of data compression.
- Section 3 presents the current (Version 0) implementation of compression at the DAAC sites, the effectiveness, and gives a justification for the compression decisions that were made.
- Section 4 explains the current and future compression techniques of data compression in industry.
- Section 5 relates data compression to the various ECS data sets and gives recommendations and justifications for data compression of the ECS data sets based the current conception of the ECS architecture.
- Section 6 gives early conclusions of compression effectiveness on ECS data.

1.3 Review and Approval

This White Paper is an informal document approved at the Office Manager level. It does not require formal Government review or approval; however, it is submitted with the intent that review and comments will be forthcoming.

This version of the white paper will serve as input to the architecture team to support decision making for the SDR time frame. This white paper will be updated as more information about the data sets, the architecture, and compression techniques become available.

Questions regarding technical information contained within this Paper should be addressed to the following ECS contacts:

- ECS Contacts
  - Howard Feil, System Engineer, (301)-925-0663, hfeil@eos.hitc.com

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Applied Information Systems, Inc.
1616A McCormick Dr.
Landover, MD 20785

2. Compression Overview & Issues

2.1 What is Data Compression (Pros & Cons)

[Excerpts from "Data Compression as a Viable Tool", Loral Aerosys]

The term data compression refers to the process of transforming a body of data (text, code, image, etc...) to a smaller representation of that same data, from which the original data stream, or some approximation of it, can be computed at a later time. Effective algorithms for data compression have been known since the early 1950's. There has traditionally been a trade-off between the benefits of employing data compression versus the computational costs incurred to perform the encoding and subsequent decoding. The advent of high-speed microprocessors and VLSI chip-sets has made data compression a standard component for use in communications, video and data storage systems for the commercial market.

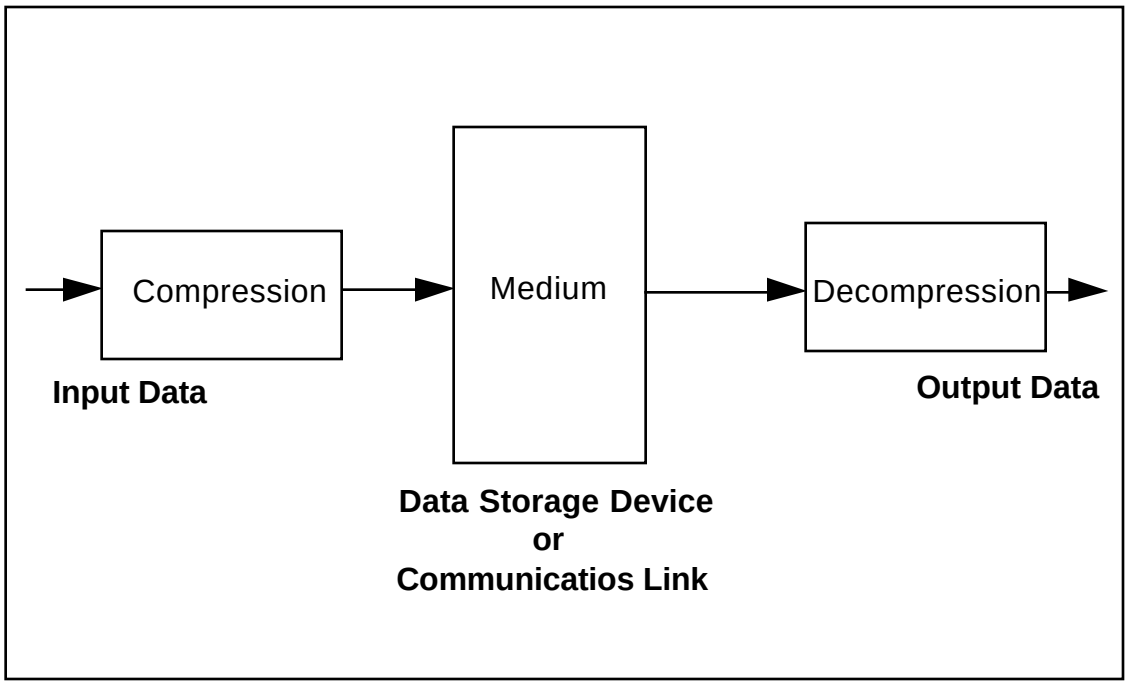
2.2 Lossy and Lossless Compression

There are two general types of data compression; "Lossy" and "Lossless". As the name implies, lossy data compression is a process where the transformation is irreversible in the sense of being able to completely recover the data as it originally existed. Lossy coding techniques arrive at a low bit rate by exploiting certain shortcomings in the human visual system, or by tolerating a degree or image distortion. As unacceptable as lossy compression technique may seem to many applications, it is still worthy of consideration since it does not necessarily destroy relevant information. In fact, techniques that are carefully chosen may actually increase the net return of information from the original data by enabling an increase in accuracy or temporal coverage as in the example of scientific data. Furthermore, lossy data compression could be ideal for browse and/or quick look applications.

Lossless data compression may be viewed as a special case of lossy compression, although in practice the techniques differ greatly. Lossless data compression is defined as the process where a transformation is fully reversible and there is no other "noise" or other artifact introduced into the original data stream.

Although data compression has many applications, the two most common areas are a) Data storage; and b) Data communications (see Figure 2-1). Data compression is by its very nature application dependent. There will not be a single solution or standard algorithm that could be recommended for all applications.

Figure 2-1 -- Data Compression Block Diagram



2.3 Data Compression Issues

In any data compression situations, there are several practical issues that play a very a important role in the choice of the compression algorithm to be used for the given circumstances, namely,

2.4 Compression efficiency

Compression efficiency is defined as the amount of bit rate reduction for a prescribed level of fidelity. Generally speaking, the compression ratio offered by lossless technique are highly data dependent. The same algorithm may have compression ratio in the area of 1.6:1 to 3:1 for 8 bits pictorial images. Compression rates for 12bit radiological images approaches 3:1. The compression ratio could be as high as 10:1 to 30:1 for documentation type of image, such as CCITT Group 4 algorithm. Arrays of floating point numbers look like white noise to most data compression schemes and compress rather poorly. Source code can normally be compressed by factors of two, but object code with its arbitrary bit patterns, does not compress well at all. FORTRAN compiled code compresses approximately 50 percent.

Lossy data compression, on the other hand, can achieve compression ratio from 8:1 to as high as 200:1. The compression ratio is usually fixed for a particular algorithm. However, different algorithms are usually needed for different data type and/or application. Typically, visual appearance and Signal to noise (SNR) ratios are used to measure the deviation of lossy data compression. However, additional processing stages after data compression could be critical for ECS browse application.

2.5 Complexity of Implementation

Data compression algorithms usually involve enormous number of multiplications, additions and comparisons per input sample. It is always a tough tradeoff between hardware or software implementation based on cost, performance and different type of applications. The ability to put significant processing power on a single chip makes sophisticated data compression algorithms truly practical. Another issue is the tradeoff between COTS, which usually complies to standards (e.g., JPEG, MPEG, CCITT, etc.), and custom implementation. In addition, a tradeoff analysis should be performed to decide where the data compression should be applied. For example: whether data compression should be performed at the host level or at the workstation and/or different storage level; whether data compression should be performed at different stage of processing data.

Furthermore, if data compression is executed in the CPU, it would be made available to application software and would be a main memory to main memory operation. This type of compression would improve I/0 channel utilization but would also increase the main memory cycles required. If invoked optionally by an application, this type of compression would not necessarily require changes to the operating system. If compression is executed in the I/0 channel, it would be visible to the operating system software and would require full message buffering in the channel controller, and the operating system would issue separate operations for compression and I/0 transfer, giving it a chance to see the size of the compressed data before it transfers the data to a device. Because of data storage requirements, this approach would probably only be useful on select channels. It does allow the device bus transfer rate to be better utilized only because compressed data moves from the I/0 channel to the device directly, but it also complicates the I/0 channel and operating system software design.

2.6 Implementation Delay

The implementation delay is the time required to encode and decode the data. Depending on the algorithms and implementation schemes, two modes of operations exist; e.g., on line and off line. An on line algorithm is real time if there exists a constant k (which does not depend on the data being processed) such that for every k units of time, exactly one new character is read by the encoder and exactly one character is written by the decoder. The only exception to this rule is that we may allow a small 'lag' between the encoder and decoder. What distinguishes this mode from an off line model is that neither the sender or the receiver can see all of the data at once; data must be constantly passing from the sender through the encoder, through the decoder, and on to the receiver. An off line model, on the contrary, is non real time. Depending on the application and data rate, one should decide whether to use on line or off line mode. For example, communication application usually requires on line mode, while storage applications may use both modes depending on other factors (e.g., costs, performance, etc.).

2.7 Sensitivity to channel (transmission and/or storage) error.

An important feature of any data compression scheme is its sensitivity to transmission and/or storage errors. Typically, more efficient data compression algorithms are more sensitive to channel errors; in many cases, one channel error could propagate over a large number of data samples, corrupting a large portion of compressed data and resulting in excessively large distortions. This nets out to the fact that a given error rate may be acceptable for the transmission or storage of

uncompressed data, but the same rate may prove to be unacceptable for compressed data. Error correction coding will not alleviate this problem in most cases, as the check bytes appended to the data stream would be removed as redundant information during the encoding process. This is one of the most difficult problem for data compression implementation, and usually takes specific techniques to address this problem. Extensive simulation runs and/or prototype system are needed to analyze this problem.

2.8 Robustness with respect to different data streams

Typically, data compression algorithm works better for a particular type of data source. Clearly text data (e.g., EOS Metadata) is quite different from the so called digitally sampled analog data such as speech, music, black and white or color video, raw and satellite imaging and non imaging data. Therefore, we need to use different data compression algorithms for different types of data. For lossy data compression, specific application in addition to data types could also play a major role to determine the data compression algorithms.

2.9 Application Impact

Each compression strategy poses a different set of problems and, consequently, the use of each strategy is restricted to applications where its inherent weaknesses present no critical problems.

2.9.1 Processing and retrieval impact

For lossy data compression, visual appearance and SNR are usually used to measure the distortion of the uncompressed data. However, additional processing could be important for certain applications. For example, tradeoff between radiometric (intensity) and geometric (spatial) distortion are important for Synthetic Aperture Radar (SAR) data.

Another potential problem is the application for data subsetting based on predefined criteria. This is especially true for lossless data compression due to the variable length block size. The utilization of special indexing schemes is one way to provide this capability.

2.9.2 Computer system integration impact

Several problems are encountered when common compression methods are integrated into computer systems, and this has prevented the widespread use of automatic data compression. For example; (1) poor runtime execution speed interferes in the attainment of very high data rates; (2) most compression techniques are not flexible enough to process different types of redundancy; and (3) blocks of compressed data that have unpredictable lengths present a storage space management problem.

When using data compression on peripherals, a significant system consideration occurs in deciding where to implement compression in the I/0 path. The compression operation inherently requires some buffering because of its rate of outputting compressed data varies widely, depending upon instantaneous compression ratios. Another potential problem is channel bandwidth. If data is compressed by a factor of 3.0:1, and data is being sent to a 4 MB/s device, then the channel must deliver a minimum of 12 MB/s. This peak rate requirement is variable depending upon compression ratio, so I/0 channel sizing cannot be calculated directly from device transfer rate alone.

Compression of information within magnetic and optical disks is practical because the internal data format on disk is generally invisible to the user and can therefore be altered to accommodate data compression. Traditionally, the operating system directly controls space allocation on the disk by keeping a map of the available space and checking each new write request against this mapping to determine when and where space is available. If compression is used in a mode transparent to the operating system, this direct control is no longer possible. To alleviate this problem, the space allocation task is moved to the disk controller.

Data compression within magnetic tape drives is well suited because space allocation is not normally involved, and data block sizes are large. The only problem that must be dealt with is that of reading backwards. Adaptive compression schemes are traditionally a one way procedure, so reversed data blocks must be stored before decompression can begin. This is a problem in start/stop mode drives versus streaming type.

2.9.3 Communication application impact

Data compression techniques that are employed to conserve communications link data must consider the intended used of the data. The user of these techniques must have the option to allocate bits to best satisfy his objective over a reasonable range of selectable options (e.g., to allocate bits in a trade off between bandwidth covered vs. quantization level). In addition, there are some general benefits to be gained from data compression in this application. Some forms of compression may perform initial steps of the final processing stream, such as in the case of Transform Coding which reduces noise artifacts prior to other processing.

Communication links pose special problems because they are traditionally noisy and increase the capacity of 'garbled data' in the decompression process along with the fact that methods employed to overcome this problem are typically error correction. As stated earlier, the redundancy found in most ECC approaches is eliminated as redundant data in the compression process. This phenomena is not found in forward error correction techniques.

3. V0 Analysis

3.1 V0 Standard Product Compression Methods

In this paper, standard products are any product which require lossless storage. Typical standard products are level 0 through level 4 products from the data pyramid.

3.1.1 Current Methods

Table 3-1 gives a summary of the compression being used at the DAAC sites. GSFC and MSFC are the only two DAACs that report most of their data is being compressed. LaRC and NSIDC also reported that a small portion of their data is being compressed. GSFC, MSFC, and LaRC reported that they were using UNIX compress (a lossless compression technique) and were achieving an average compression ratio of about 2:1 across the data they were compressing. NSIDC said they were using GNUZip (a lossless compression technique) on a small portion of their data because they found it to be a little better then UNIX compress.

The DAAC sites using compression consider the CPU time to compress the data negligible, and most offered to distribute the data in a compressed or uncompressed format. The DAAC sites encourage distribution of data in the same format as it was stored in on the DAAC sites.

No DAAC site reported complaints or problems with their current compression or storage techniques.

Table 3-1, V0 System Compression

Location	Standard Product	Distribution	Browse Product	Dist. of Browse
ASF - Fairbanks, AK	With the exception of one Level 2 product which is being compressed with run length encoding (RLE), no compression is being used at ASF.	N/A	N/A	Hard Copy File Folder
EDC - Sioux falls, SD	No compression is used on standard products.	Compression Option. The user has the option of selecting a switch in the IMS toolbox which will compress files on their local workstation using a home grown algorithm called Encode 76.	Granules are decimated and then stored using the JPEG standard DCT. Browse products are visually lossless with a quality factor of 30 yielding a compression ratio of 10:1.	Individual granules are fetched and put together by the IMS into an HDF format before distribution to the user. In the future will store browse images in an HDF format.

GSFC - Greenbelt, MD	Some products are compressed using UNIX Compress. Data that was found to be compressible is compressed (AVHRR, TOVS3, CZCS), and data that was found not to be compressible was left uncompressed (UARS).	Encouraged to receive in compressed format. In general, data is distributed in the same format that it is stored in.	Browse products compressed using UNIX compress.	Viewed with a browsing tool.
JPL - Pasadena, CA	No, the current volumes are not enough to justify using compression.	N/A	No compression used on browse products.	Viewed with a browsing tool.
LaRC - Langley, VA	UNIX Compression is being applied to Level 0 and Ephemeris data only.	Level 0 and Ephemeris data not available to clients. Available products are not compressed.	Browse products are stored in and HDF format with the JPEG compression applied to the granules. Compressed 9:1.	Viewed using a HDF utility developed at NCSA.
MSFC - Huntsville, AL	Most data sets are compressed with UNIX Compressed.	User can get compressed or uncompressed data for most data sets. Large granules only distributed in a compressed format.	No compression used on browse products.	N/A
NSIDC - Denver, CO	GNU Zip used only for SSM/I and TOVS3 data set.	Not distributed in a compressed format because of the burden it places on the end user.	No compression used on browse products.	N/A

3.1.2 Decision factors

UNIX compress utilizes the Lempel-Ziv-Welch (LZW) algorithm for compression (See Section 4.2.5 for further discussion of the LZW the algorithm.) The same algorithm is used in the and the VMS compression utility. The major reason UNIX compressed was used at the DAAC sites using compression was he popularity and portability of UNIX compress. The compress algorithm is available on almost all UNIX, VMS platforms, and is even available on PCs. Although UNIX compress does not get a top compression ratio in most cases, there is no other compression algorithm that will always be better. There are compression algorithms which are better then UNIX compress most of the time on image data, but these are not readily available on multiple platforms.

NSIDC chose to use GNU Zip which is only a slight variation of Lempel-Ziv 1977 (LZ1) and only gives slightly better compression. LZ1 gives better compression then LZW but at the cost of a greater CPU burden. GNU Zip is readily available on multiple platforms, and its source code is freely distributed. GNU Zip is not as popular as UNIX compress largely because UNIX compress is the default package built into almost every UNIX box today and requires no effort for the user to utilize. For the data sets NSIDC is applying GNU zip to a typical compress ratio of 2:1 was achieved.

Most DAACs chose not to use compression at all or only on some data sets. There were several fundamental reasons. One of the biggest reasons was that compression was not needed given their current data volumes and their current storage capacity. Using a compression algorithm only created unnecessary overhead for the DAAC to perform effectively. Some of the DAAC sites reported that their data was not very compressible only getting a 20 to 30% gain. Level 1B data is already highly compacted and no compression method will reduce its size very much. The DAACs felt that 20 to 30% compression was not enough to justify compression.

Another reason compression was not used was that compression created a significant burden on the end user to decompress the data. If the compressed granules were delivered to the client, decompression of large granules or a large number of granules could take a considerable amount of time.

3.1.3 Effectiveness

The UNIX compress algorithm uses the LZW algorithm which is believed to yield compression ratios typically in the 1.3:1 to 2:1 range for Landsat and AVHRR data. For some data sets (UARS) it actually expanded the data set; hence, it was not used to compress these data sets. Most DAAC sites reported that compression varied significantly from data set to data set and even from granule to granule.

When questioned, most DAAC sites reported the average compression ratio for the data they were compressing was 2:1. Most DAACs reported performing small studies to reach this conclusion. However, it is not known if the studies only included "interesting" data, and it is not known of what portion of the data is "interesting". Interesting data is data over land of cities, cloud formations etc. Most of the planet is not interesting most of the time. That is, most of the planet is water and not cloudy. It is possible that some data sets are more compressible then 2:1 if they are designed to only take meaning results over "interesting" formations. It seems reasonable to assume that Landsat pictures over the ocean should be more compressible compared to Landsat images of Washington D. C. One may questions the initial DAAC survey results of overall compression of 2:1; however at this time, there are no more detailed analysis of global satellite data compression effectiveness available.

The DAAC sites using UNIX compress reported that the CPU time to compress/decompress granules was considered negligible, and the CPU time for the PGS to decompress the data before applying an algorithm was also negligible. However, on a workstation class machine, it can take considerable time to decompress large granules. A full granule of Landsat TM data on a SPARC 2 station could take approximately 20 minutes to decompress.

GNU zip tends to give better compression ratios then UNIX Compress but at a greater CPU cost. GNU zip is only a slight modification of the Lempel-Ziv 1977 algorithm. As with most issues in

compression today, for every GNU zip user who says the algorithm is better, another user will claim it is no better or even worse.

Lower level products such as Level 1B are hard to compress because the data is very compacted. Higher level gridded products such as Level 3 data are easier to compress because of the formatting that has been added to the data. EOSDIS has a large variety of data formats and no general rule about compression can be made. Some data sets and products are highly compressible while other can be very hard to compress. (See section 5.2 for further discussion.)

3.2 V0 Browse Product Compression Methods

Table 3-1 shows a summary of compression techniques at the DAAC sites for browse products. In this paper, browse products are defined as lossy products that are representative of standard products suitable to meet the requirements set in the requirement specification.

It is difficult to design a browse product because assumptions have to be made about the quality of the product. What may be acceptable to one researcher maybe highly unacceptable to another scientist.

3.2.1 Current Methods

Browse products are currently created by sub sampling the full size granules by factors typically ranging between 16:1 to 48:1. Sub sampling the original greatly reduces the size and volume of the browse file. As shown in table 3-1, most DAAC sites perform no further processing of the browse images.

EDC and LaRC were the only two DAAC sites that reported using a compression technique on browse product which can be further compressed in a lossy nature. LaRC reported that they had incorporated compression of their browse products with HDF, and EDC plans to do so in the near future.

Both EDC and LaRC are using the current JPEG lossy standard which uses the Discrete Cosine Transform (DCT). See section 4.3.1.1 for further discussion of the JPEG lossy standard.

3.2.2 Decision Factors

The JPEG standard was chosen for many of the same reasons UNIX Compress was chosen. JPEG viewers are widely available and in the public domain on almost every computer system, and JPEG encoder's source codes are readily available and can be easily tailored to a dataset. The current JPEG standard is also highly effective, and although there are other lossy compression methods which can produce better results, there is no other lossy compression method which gives significantly better quality/compression ratio and at a faster speed.

3.2.3 Effectiveness

The JPEG standard is highly effective and easy to implement. There is little development overhead associated with implementing JPEG because of the large quantity of code available today in the public domain. JPEG does not require the development of any code books or data set constants.

The DAAC sites that implemented JPEG researched with the scientist to determine what an acceptable level of loss would be in the browse images. The scientists required visually lossless

data (i.e. the human eye could not perceive a difference). The resulting JPEG implementation had a JPEG quality factor (JPEG quality factor is an internal use constant to the algorithm) of 30 resulting in a typical compression ratio of 10:1.

The DAAC sites reported that scientists using JPEG had no complaints with the browse images. The time it took the viewer to decompress the image was well offset by the reduction in data volume. The JPEG viewers work fairly fast. A typical decompression time is only a few seconds.

4. Types of Data Compression Algorithms

4.1 Basic Data Compression Facts

[From Chiang *et al.,* 1991]

All commonly used data compression schemes are based on the concept of entropy coding and redundancy reduction, that provide for the following;

- Random data cannot be compressed, and if processed will actually expand in size and slow system speed.
- Data that has been compressed by an optimal scheme cannot be compressed further, and the output of this process will be a random data stream
- If data compression is implemented at the 'system' level, no benefit will be derived by incorporating additional data compression in a peripheral or communication channel

Four types of redundancy can be found in most commercial and scientific data;

- Character Distribution: In a typical character string, some characters are used more frequently than others e.g. vowels
- Character Repetition: When a string of repetitions of a single character occurs, the message can usually be encoded more compactly that by just repeating the character symbol.
- High Usage Patterns: Certain sequences of characters will reappear with relatively high frequency and can therefore be represented with relatively fewer bits for a net saving in time and storage space.
- This is also commonly found in imagery positional redundancy. If certain characters appear consistently at a predictable place in each block of data, they are at least partially redundant.

These four types of redundancy obviously overlap to some extent. By examining the type of redundancy, some decisions regarding the choice of a compression strategy can be made;

- The redundancy type found in a certain application is important, but the predictability of redundancy type is also just as important.
- An adaptive type of compression would be of little benefit for applications with predictable redundancy such as text, but would be valuable for business files or scientific data. Any compression method that adapts to the statistics of the subject data and has a finite implementation, will have compression effectiveness sensitive to message length.
- The length of the data being transmitted has some importance because adaptive techniques are awkward or ineffective on short messages. Short blocks are penalized by a start up overhead needed to convey subject statistics. Large blocks suffer a loss in efficiency because the blocks lack stable statistics, a typical occurrence in commercial computer data.
- System data transfer rates determine if a one pass procedure is needed for speed or if the greater overhead of a two pass approach can be justified by better compression results.

In reviewing these criteria, please note that, much of the published theory in the data compression field assumes arbitrarily large translation tables and ergotic data sources, neither of which occur in practice.

4.2 Lossless Data Compression Algorithms

The basic types of lossless data compression techniques revolve around the concept of entropy reduction and redundancy reduction.

4.2.1 Huffman and Shannon Fano Algorithms

[Portions From Chiang *et al.,* 1991]

Huffman coding is the most well know technique for data compression, dating back to 1952. Huffman coding translates fixed size packets of input data into variable length symbols. The standard Huffman encoding procedure prescribes a way to assign codes to input symbols such that each code length in bits is approximately Log_2(Symbol probability). Symbol probability is the relative frequency of occurrence of a given symbol (expressed as a probability).

Problems Encountered with Huffman Coding:

- The size of the input symbols is limited by the size of the translation table needed for compression.
- With Huffman encoding the complexity of the decompress process is big. The length of each code must be interpreted for decompression and is not known until the first few bits are interpreted.
- You need to know the frequency distribution for the ensemble of possible input symbols. The distributions for data files, however, are very application specific and files such as object code, source code, and system tables will have dissimilar characteristic distributions.
- The optimized code will only exhibit efficient performance over a narrow range of data entropies.

Dynamic Huffman Codes. In order to eliminate the two pass approach (first to gather statistics, then to code) a dynamic approach is taken to make Huffman an on line method. Each time the encoder sees an input element it increments its counter by one, and also builds a new tier. Through deterministic methods a tier of all possible elements is built.

Shannon Fano is similar in flavor to Huffman, but that the tier construction algorithm works from "top down", instead of 'bottom up' as in Huffman.

4.2.2 Binary Arithmetic Coding (IDRC, BAC , Q Coder, WNC)

[Portions from Chiang *et al.,* 1991]

Arithmetic coding, which is an enhancement of Huffman coding, is a data compression technique that encodes a data string by creating a code string which represents a fractional value of the number line between 0 and 1. The coding algorithm is symbolwise recursive i.e. it operates upon and encodes/decodes one data symbol per iteration or recursion. On each recursion, the algorithm successively partitions an interval of the number line between 0 and 1, and retains one of the partitions as the new interval. Thus, the algorithm successively deals with smaller intervals, and the

code string viewed as a magnitude, lies in each of the nested intervals. The data string is recovered by using magnitude comparisons on the code string to recreate how the encoder must have successively partitioned and retained each nested subinterval.

Arithmetic coding maps a string of data symbols to a code string in such a way that the original data can be completely recovered from the code string. The encoding and decoding algorithms perform arithmetic operations on the code string. One recursion of the algorithm handles one data symbol. Arithmetic coding is actually a family of codes which share the property of treating the code string as a magnitude.

All data compression approaches have a model which makes some assumptions about the data and the events encoded. The first order Markov model is a dependent model , where we have a different expectation for each symbol depending on its context. The purpose of a context is to provide a probability distribution or statistics for encoding the next symbol.

Data compression results from encoding the more frequent symbols with short code string length increases, and encoding the less frequent events with the long code length increases. Arithmetic coding is capable of accepting successive events from different probability distributions, and acts directly on the probabilities and can also adapt 'on the fly' to changing statistics. It also operates in a FIFO fashion.

Each codeword (code point) is the sum of the probabilities of the preceding symbols. The width or size of the subinterval to the right of each code point corresponds to the probability of the symbol. In arithmetic coding we treat the code points, which delimit the interval partitions as magnitudes. Arithmetic coding is capable of using arbitrary probabilities by keeping the product to a fixed number of bits of precision. A key advantage of arithmetic coding over other methods is that it contains the required precision so that significant digits of the multiplications do not grow with the code string.

For arithmetic codes we can view the code as mapping a data string to an interval of the unit interval.

The BAC algorithm, is a special case of arithmetic coding, and may be used for encoding any set of events, whatever the original form, by breaking the events down for encoding into a succession of binary events. The BAC accepts this succession of events and delivers successive bits of the code string.

Arithmetic codes generate the code string by adding a summand to the current code string and shifting the result. This summation operation creates a situation called carry over. The basic conceptual input/output view of the algorithm both for the compression and decompression process contains an encoder and a special arbitrarily long FIFO buffer "Q" (the "Q" Coder) which handles the code string and the carry over . The encoder and decoder in practice are interfaced to the original data via a statistics unit which provides the skew numbers.

Arithmetic codes can achieve compression as close to ideal as desired for given statistics. The Q Coder is a new development in arithmetic coding. It combines a simple but efficient arithmetic approximation for the multiply operation, a new formalism which yields optimally efficient hardware and software implementations, and a new technique for estimating symbol properties of any method known. The compression process is divided into three basic parts; (1) a model which converts uncompressed data into binary decisions, (2) a probability estimator and (3) an arithmetic coder. The decompression process has three similar parts; (1) an arithmetic decoder, (2) the same

probability estimator, and (3) and the inverse model which converts the binary decisions back into uncompressed data. The Q Coder is the heart of the data compression scheme used by IBM in its IRDC.

Currently the best arithmetic results tend to be coming from the Witten-Neal-Cleary (Witten *et al.,* 1987) encoder.

An issue of concern with arithmetic coding is that some of these algorithms are patented and copyrighted. These algorithms tend to be fairly effective generic coders, but they can also be specialized for a particular data or for a given set of statistics.

4.2.3 Adaptive and Predictive Coding

[Portions From Chiang *et al.*, 1991]

Adaptive and predictive coding methods can be added to the above algorithms to specialize them for an application such as image encoding. Predictive and adaptive coding will increase the compression ratio particularly when predictions about the data can be made such as with image data. Of course, this added compression will come at a greater CPU cost at compression and decompression.

The application of an adaptive binary arithmetic coder, such as the Q coder, to raw bit plane data is straightforward. Langdon developed a context model whereby neighboring pixels in a bit plane form a context, similar in form to the structure of a prediction set used in predictive coding. These neighbors form a binary word that becomes the context address of the Q Coder. More pixels allow for a greater number of context possibilities. An extension of the context pixel assignment is to form the context based on pixels from the current and previous bit planes.

By applying the Q coder in connection with the 2-D autoadaptive coding, the decisions are entropy encoded via the Q Coder to form a binarization tree, and eliminates the inefficiencies present in autoadaptive code when starting with too large a block size.

Similar techniques can be used to develop predictors for a multispectral image. Instead of just using the nearest neighboring pixels to form a predictor as above in the 2-D case, the predictor can also use the pixels in the adjacent bands. Band ordering techniques can also be employed to attempt to use the most related bands as predictors for each other. [Tate, DDC94]

Differential pulse code modulation (DPCM) is a popular prediction technique where the encoder attempts to encode only the new information between pixels. This technique is aimed specifically at remove the mutual redundancy between successive pixels. DPCM can be adapted in a similar method as the arithmetic context based Q-Coder discussed above to attempt to exploit similarities in the 2-D case and in the multispectral case.

4.2.4 JPEG Lossless Compression

[Portions from Tilton *et al.,* 1991]

To facilitate the exchange of compressed data, certain, the committee known as the Joint Photographic Experts Group (JPEG) worked for several years to establish compression standards for still continuous tone images (Wallace, 1991; Pennebaker & Mitchell, 1993). JPEG compression standards include two basic compression methods: a predictive method for lossless compression and a lossy compression approach based on the Discrete Cosine Transform (DCT)

(See section 4.3.1.1 for JPEG lossy standard.) One advantage of using this standard compression approaches is that very efficient and cost effective software and hardware implementations are becoming available commercially.

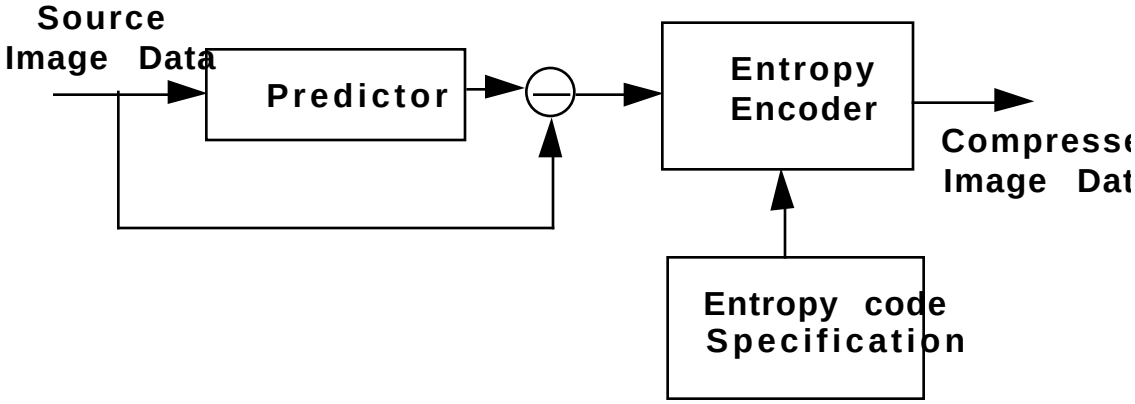


Figure 4.2.4-1 JPEG Lossless encoder flow Diagram

The main processing elements of the JPEG lossless compression technique are a predictor, entropy encoder for prediction error, and entropy code specifier. These main blocks are illustrated in Figure 4.2.4-1. The JPEG standard provides a choice of eight different predictors (see Table 4.2.4-1) using a choice or combination of three spatially adjacent neighboring samples as listed in Table 4.2.4-1. The entropy encoder (a class of lossless compression algorithms based on information theory considerations) is used to compress the prediction errors. The baseline entropy encoder is the Huffman encoder (Huffman, 1952). An option is given for using an a variation of the arithmetic encoder Witten-Neal-Clearly (WNC) (Pennebaker & Mitchell, 1993) instead.

Table 4.2.4-1 JPEG Predictor selection table

Selection Value #	Prediction Location for x
1	a
2	b
3	c
4	a+b-c
5	a+(b-c)/2
6	b+(a-c)/2
7	(a+b)/2

	c	b
	a	x

The JPEG lossless standard with the WNC encoder achieves one of the highest compression ratios for image data; however, it is also has greater CPU demand then most of the other algorithms being discussed here.

See end of section 4.3.1.1 for discussion of a new JPEG standard.

4.2.5 The Lempel Ziv Series (LZ1, LZ2, DCLZ, LZW) (UNIX Compress, PKZip)

[Portions From Chiang *et al.*, 1991]

In the late '70's, researchers Lempel and Ziv fathered a series of adaptive string based data compression algorithms for data. Their algorithms known today as LZ1 (LZ77), LZ2 (LZ78), and LZW are able to achieve high compression rates on many types of data.

They propose to evaluate the complexity of a finite sequence from the point of view of a simple self delimiting learning machine which, as it scans a given n digit sequence from left to right adds a new word to its memory every time it discovers a substring of consecutive digits not previously encountered. The size of the compiled vocabulary and the rate at which new words are encountered along the way serve as basic ingredients in the proposed evaluation of the complexity of the sequence.

Their compression algorithms are an adaptation of a simple copying procedure discussed in their studies on the complexity of finite sequences. Basically, they employ the concept of encoding future segments of the source output via maximum length copying from a buffer containing the recent past output. The transmitted codeword consists of the buffer address and the length of the copied segment. With a predetermined initial load of the buffer and the information contained in the codewords, the source data can be readily be reconstructed at the decoding end of the process. The main drawback of their algorithms is their susceptibility to error propagation in the event of a channel error.

Their compression algorithms consist of a rule for parsing strings of symbols from a finite alphabet into substrings or words whose lengths do not exceed a prescribed integer, and a coding scheme which maps these substrings sequentially into uniquely decipherable codewords of fixed length over the same alphabet. Conceptually, the sliding dictionary method is perhaps the simplest method that employs a dynamically changing dictionary. The LZ1 algorithm, which can be shown to be perfect in the information theoretic sense, works as follows: At each stage, the longest prefix of the unread portion of the input stream that matches a substring of the input already seen is identified as the current match. A triple is then transmitted, and the input is then advanced past the current match and the character following the current match. Thus, the LZ1 algorithm remembers the entire input string and hence uses a dictionary that is unbounded in size. There are no bound on the number of bits needed to encode the triples that are transmitted and so a variable length coding scheme is needed. The sliding dictionary method, LZ1, uses fixed size pointers; instead of remembering the entire input stream, it remembers only a fixed number of characters back, and instead of pointer guaranteed progress, it uses dictionary guaranteed progress by the reserving of codes for the characters.

The GNU Zip utility and the PKZip utility use the LZ1 algorithm.

The dynamic dictionary method is the implementation of Lempel and Ziv's second universal data compression algorithm and is called LZ2 (LZ78). The LZ2 algorithm, like LZ1 can be shown to be perfect in the information theoretic sense. At each stage, the longest prefix of the input stream that matches one of the strings in the local dictionary is identified. Then a pair is transmitted, where 'p' is a pointer to 't' in the dictionary and 'c' is the next input character following the current match. As with LZ1, the transmission of 'c' is a pointer guaranteed process. The input is then advanced past 't 'and 'c' and the string 'tc' is added to the local dictionary. Thus, the local dictionary used by

the LZ2 algorithm is unbounded in size and like LZ1, a variable length coding scheme must be used to construct pointers.

The DCLZ algorithm is fundamentally the same as the LZ2 algorithm. The basic advantage of the DCLZ algorithm is that the dictionary is embedded in the codewords and is not explicitly transferred with the compressed data as a library of codes. This is accomplished by synchronizing the compression/decompression process such that during decompression an identical dictionary can be built to match the dictionary built during compression.

The LZ2 and DCLZ algorithm tend to be more popular for implementation in hardware because unlike the LZ1 algorithm they do not require the storage of the entire input stream to encode the next character. Hence, LZ2 and DCLZ require less memory, but are not as optimal as LZ1 particularly if a limit is put on the dictionary size.

Welch, who experimented with the LZ2 algorithm and applied the FC Freeze heuristic, calls it the LZW algorithm. The LZW algorithm is organized around a translation table, referred to as a string table, that maps strings of input characters into fixed length codes. The LZW string table has a prefix property in that for every string in the table its prefix string is also in the table. The LZW string table contains strings that have been encountered previously in the message being compressed. LZW uses the 'greedy' parsing algorithm, where the input string is examined character serially in one pass, and the longest recognized input string is parsed off each time. A recognized string is one that exists in the string table. The strings added to the string table are determined by this parsing: Each parsed input string extended by its next input character forms a new string added to the string table.

This algorithm makes no real attempt to optimally select strings for the string table, or optimally parse the input data. It simply produces compression results, that, while less than optimum, are effective. Since the algorithm is quite simple, its implementation can be very fast. The principle concern in implementation is storing the string table. This form is well suite for hashing methods and some type of hashing is normally used. Unfortunately, this simple algorithm has two complicating problems, (1) It generates the characters within each string in reverse order, (2) It does not work for an abnormal case.

The principal implementation decision is choosing the hashing strategy for the compression device. Software implementation of LZW is possible but significantly slower. Compression speed is very sensitive to the hash calculation time in the inner loop. Software hashing is relatively slower and less effective than hardware hashing, and generally needs to be hand coded in machine language.

UNIX compress and VMS compress use the LZW algorithm. These utilities tend to be faster but not as effective as GNU Zip and PKZIP.

The Lempel-Ziv algorithm is patented algorithm and some of the variations describe here also have copyrights associated with them. Despite the some criticism in industry about the Lempel Ziv algorithm's effectiveness there is no algorithm that consistently works better in all cases.

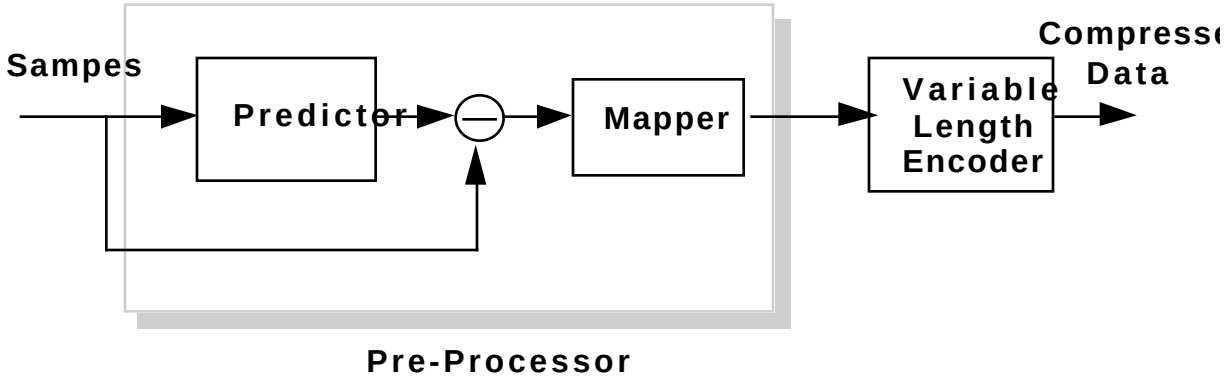
4.2.6 Rice Algorithm

[Portions from Tilton *et al.*, 1993]

The Rice algorithm (Rice *et al.,* 1991) is a lossless compression scheme that can adapt to data of any entropy range *(i. e.,* level of information content). In general, the Rice algorithm will compress presentation image data better then Lempel-Ziv.

The algorithm consists of two separate functional parts (see Figure 4.2.6-1): a pre-processor and variable length encoder. The pre-processor consists of a predictor followed by a symbol mapper, while the variable length encoder performs the actual adaptive symbol encoding.

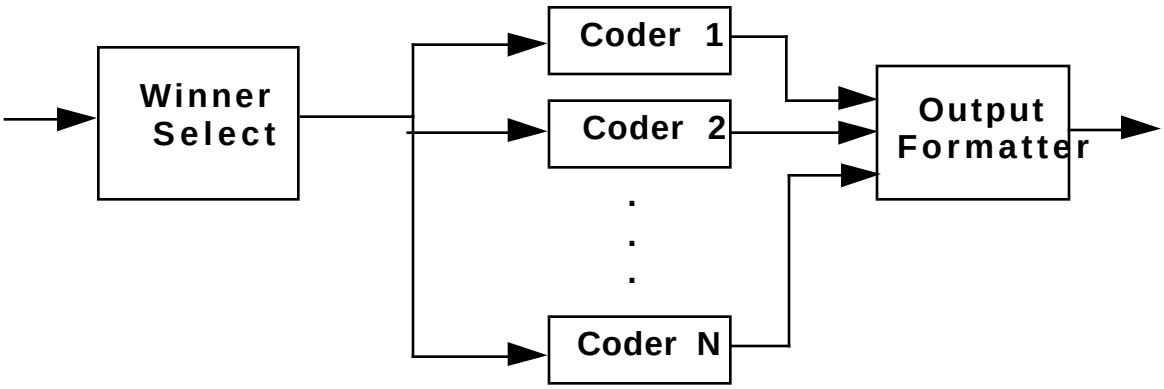
Figure 4.2.6-1 Rice encoder flow diagram



The pre-processor spatially decorrelates the data through a prediction scheme *(i. e.,* the prediction errors are much less spatially correlated than the original data), and maps the resulting prediction errors to a sequence of non-negative integers. While the baseline Rice algorithm employs a simple sequential difference predictor, higher order predictors can be used. The degree of spatial correlation exploited by the Rice algorithm depends upon the order of the predictor employed.

The variable length encoder, described graphically in Figure 4.2.6-2, consists of multiple coders, each targeted for a different source entropy level. The entropy coder begins with a winner selection this allow for sharing of hardware among the many parallel coders. If the encoder cannot compress the data, the default coder sends out ID bits followed by uncompressed data.

Figure 4.2.6-2 Rice Variable Length Entropy Encoder



The Rice algorithm exploits the spatial correlations in image data and generally provides better compression image data when compared to text oriented compression algorithms (such as Lempel-Ziv, see section 4.2.5). A related approach including both prediction and error modeling and various refinements to increase algorithm efficiency, was recently reported by Howard and Vitter (1993). Howard and Vitter report their modifications of Rice make it 5 times faster then lossless JPEG with about the same compression ratio.

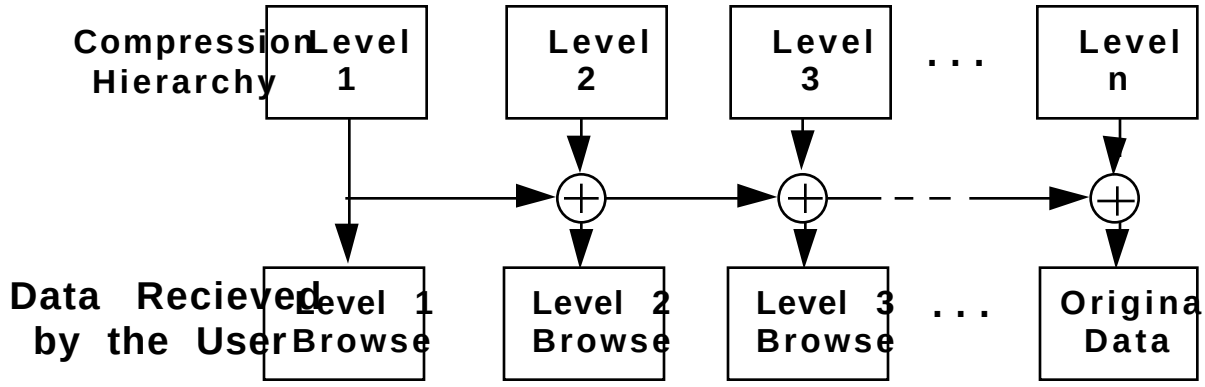
COTS compress chips implementing the Rice algorithm are available today. Some companies claim their Rice Chips will go directly onto a SCSI personality card making the compression transparent to the system. The Rice algorithm has been used successfully on board spacecraft by NASA to reduce the volume of SAR data transmitted by the spacecraft. NASA Goddard has also developed Rice chip sets.

4.2.7 Progressive Techniques

Progressive techniques combine the methods discussed in section 4.3 Lossy Data Compression Algorithms with a residual image. The lossy image(s) and the residual are also compressed using one of the above techniques. If the compression techniques are well tuned to complement each other in an ideal case, total compression may be increased by 10-20%.

A progressive technique enables a client to view a browse image before getting the final product, and enable the final product to be sent with no redundant data being sent to the client. Some implementations of progressive techniques allow the client to get better and better image qualities without sending redundant data. The user can stop the refinement process after any stage of refinement, if so desired, saving data transmission costs. Figure 4.2.7-1 gives a graphical representation of a progressive technique implementation.

Figure 4.2.7-1 General Progressive Technique



Progressive techniques save data storage and transmission cost but significantly increase compression and decompression complexity. Not all lossy techniques naturally lend themselves to multi-level transmissions, but any lossy technique can have a residual stored and sent to make the final data lossless. A big disadvantage of progressive techniques is that the lossless product is not readily accessible for processing and has to be built by combining several files. If each of these files is compressed using an entropy encoder, this could add significant processing time.

4.2.8 Other Lossless Compression Techniques

Other techniques include one dimensional run length encoding, bit plane processing, and binary compression.

One dimensional run length encoding: The application of 1 D run length encoding to bit planes is well known, and has appeared most recently in the field of facsimile. In this method groups of 1's and 0's are lumped together in an abbreviate mnemonic which can be losslessly unencoded.

Bit Plane Encoding: An NxM bit image can be considered as a set of K NxM 1 bit planes, each of which can be coded via a standard binary compression technique. Common bit plane encoding algorithms involve the use of run lengths of the 1's and 0's, or two-dimensional variable block coding.

Binary compression techniques that involve the formation of one dimensional run lengths or two dimension variable size blocks work best on coherent bit planes, that is, long runs or large uniform areas of 1's and 0's. The Gray code is a method of encoding a set of numbers such that the successive numerical changes will result in a change of only one bit in its binary representation. The hardware implementation of the Gray code is simple and inexpensive.

4.3 Lossy Data Compression Algorithms

Lossy coding techniques to arrive at low bit rates by exploiting certain shortcomings in the human visual system, or by tolerating a degree of image distortion. Visually lossless is a term used to describe a lossy coding technique whereby the reconstructed image is visually indistinguishable from the original under specified viewing conditions. Many techniques are commonly used such as: DCT, DPCM and Vector Quantization.

4.3.1 Transforms

A significant amount of work in lossy image compression revolves around the use of transforms or block quantization. A block of data is unitarily transformed so that a large fraction of the total

energy is packed into relatively few transform coefficients. The coefficients can then be quantized and encoded. An optimum transform coder is defined as the one that minimizes the mean square distortion of the reproduced data for a given number of total bits. The most popular transform today is the Discrete Cosine Transform (DCT) described below. Other transforms include the Karhunen-Loève Transform (KLT), Subband/Wavelet, Sine, DFT, Hadamard, HAAR, Slant, and Fractals.

All transforms revolve around the same concept of condensing energy into a smaller number of coefficients. For this reason the below discussion will only discuss the three most popular: the DCT, the KLT, and Subband/Wavelet. Some applications revolving around these transforms will also be discussed.

In general, transform coding achieves relatively larger compression than predictive methods. Any distortion due to quantization gets distributed over the entire image which creates a more visually lossless appearance then with predictive methods. At low distortion levels predictive and transform methods will perform very similarly, but for higher acceptable distortion levels transforms will out perform predictive methods. This is because of two reasons. First, predictive coding is quite sensitive to changes in the statistics of the data and second because a finite-order causal representation of a two-dimensional random field may not exist. However, predictive methods are significantly less computationally expensive then transforms.

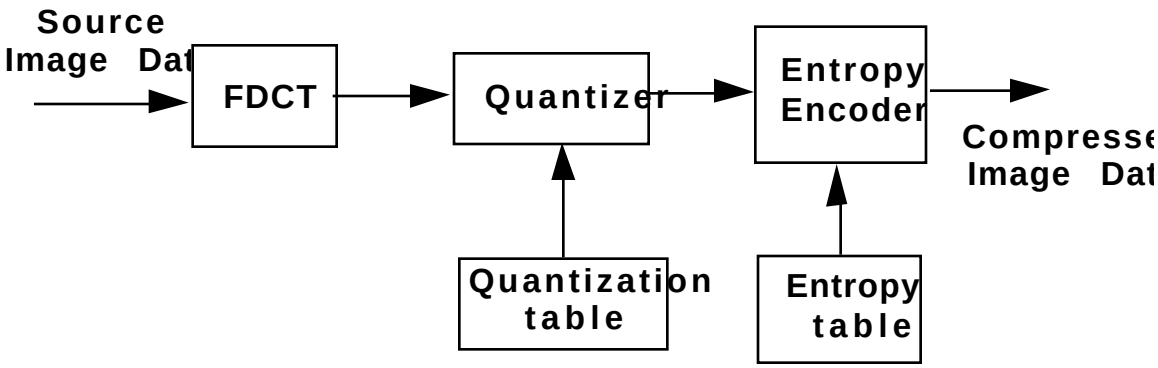
4.3.1.1 JPEG Lossy/Discrete Cosine Transform (DCT)

[Portions from Tilton *et al.,* 1993]

The JPEG lossy compression approach is based on the Discrete Cosine Transform (DCT) of 8 x 8 blocks from the input image. In the encoding process, each 8 x 8 block is transformed by the forward DCT (FDCT) into a set of 64 DCT coefficients (see Figure 4.3.1.1-1). These 64 coefficients are then normalized and quantized by dividing with a user defined 64 element normalization array and rounding to the nearest integer. Finally, the quantized coefficients are encoded with an entropy encoder.

The first DCT coefficient is called the zero frequency or DC coefficient, and is entropy encoded in a different manner than the remaining 63 DCT coefficients, called the AC coefficients. Before entropy encoding, the quantized DC coefficient of each block is replaced by the difference between it and the quantized DC coefficient from the previous block. The quantized AC coefficients from each block are converted to a one-dimensional vector through a zigzag scan before entropy encoding. Finally, the DC and AC coefficients are entropy encoded using separate entropy tables.

Figure 4.3.1.1-1 JPEG/DCT Lossy Image encoding scheme



The compressed output stream for each 8 x 8 block consists of the encoded DC coefficient followed by the encoded AC coefficients. The decoding process is a straightforward reversal of the encoding process. The 64 coefficients are decoded and used to reconstruction 8 x 8 coefficient image which then is mapped back to image space by Inverse DCT (IDCT).

In some implementations of the JPEG/DCT lossy compression, a quality factor *(e. g.,* Q) is used to describe the degree of quantization employed in the quantization step. In some implementation, Q can properly range from 25 to 100 (a Q of less than 25 is possible, but a JPEG conforming decoder would not be able to decode the result). A Q of 100 refers to no quantization, with any resulting compression loss coming solely from subsampling or roundoff errors. A Q value of 100 gives relatively low compression ratios. A Q of 50 is quantization table scale factor 1.0 (unmodified JPEG table i.e. default). For most image data the value of Q produces a reasonably high compression ratio with good reconstructed image quality. A Q of 75 is quantization table scale factor 0.5 (one-half the standard quantization), and produces moderate compression with visually high quality image reconstructions. A Q of 25 is quantization table scale factor 2.0 (twice the standard quantization). Most images will appear blocky at this level of quantization. However, a much higher compression ratio can be achieved.

The baseline JPEG/DCT accepts 8-bit images and uses separate Huffman tables (Huffman, 1952) for entropy encoding the quantized DC and AC coefficients. However, the options in the JPEG lossy standards allow either 8-bit or 12-bit precision with either Huffman or arithmetic encoding (Witten *et al.,* 1987) of the coefficients. Other options exist in the JPEG standard for the special case of RGB color image data, and for progressive and hierarchical compression modes. The initial image reconstruction for the progressive mode is a full size low spatial resolution version of the image, while for the hierarchical mode it is a reduced sized version of the image. More refined reconstructions are produced in stages for both the progressive and hierarchical modes.

The implementation of DCT can be computational intensive, there are techniques such as Hadamand transformation which can simplify the implementation complexity. Currently, IMOS has COTS chips can perform 16xl6 DCT with clock rate of 20 MHz.

Currently the JPEG standard only exists for 8 or 12 bits per pixel. It is not trivial to develop a standard that will work with more bits per pixel. However, this development cost would be incurred for implementation any of the other transforms discussed in this paper unless previous research implementing the transform can be found.

As of early 1994, JPEG put out a call for paper to implement a new JPEG standard both lossless and lossy. The international committee is also considering a multispectral standard and a satellite image standard. The international committee stated that no member on the committee and the level

of expertise needed to write a satellite image standard and this issue was referred the U.S. committee. In the next few years, it is highly likely that JPEG will release a new entirely updated standard, and a multispectral and satellite image standard which would be a new addition.

The Independent JPEG Group's free JPEG implementation, version 1, release date October 7, 1991. Version 4 was released on December 10, 1992, and it (or a later version) is available by lnternet anonymous FTP from ftp.uu.net graphicsfjpeg/jpegsrc.v4.tar.Z.

4.3.1.2 Karhunen-Loève Transform and other Transforms

The implementation of the KLT and most of the other transform methods would be very similar to the method described above in the DCT. Each transform needs to tweaked to develop a good quantization table for encoding purposes. An even better quantization and encoding method can be developed if allot of information is known about the kind of images that will be transformed.

The KLT is considered an optimum transform coder because it minimizes the mean square distortion of the reproduced data for a given number of bits. For this reason, the KLT is often referred to as the baseline standard which all other methods can be compared to. However, the KLT can be difficult to implement and very computationally costly. For these reasons, the cosine or other sinusoidal transforms are preferable.

Because most or the basic concepts of implementing all transforms is very similar, only a handful will be discussed in this paper in great detail. Below is a brief explanation of some of the advantages of the other transforms.

Cosine:	Perform best for highly correlated data.
Sine:	Useful for implementing a fast KL or recursive block coder.
Hadamard:	Useful for small block sizes (4 x 4). Simpler then sinusoidal.
Haar:	Useful if higher spatial frequencies are to be emphasized. Poor compression on a mean square basis.
KL:	Optimum on mean square basis. Difficult to implement. Sinusoidal transforms are preferable.
Slant	Best performance among non sinusoidal fast transform.
Fractal	Good for natural looking images.

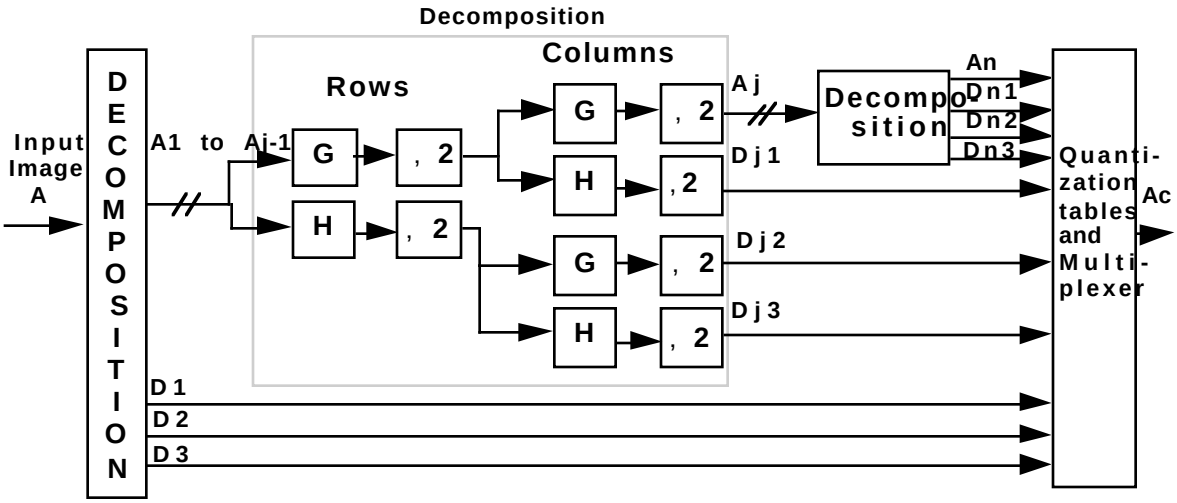
4.3.1.3 Subband/Wavelet Compression

[Portions from Tilton *et al.,* 1993]

The related methods of subband and wavelet-based image compression have been much studied as of late. In these methods, images are decomposed into a number of frequency bands using "analysis filters" such as quadrature mirror filters, or wavelet filters. For two-dimensional image data, four frequency bands are most commonly employed, with the first band containing the horizontal and vertical low-pass filter result. The other bands then contain the other results from combining horizontal and vertical low- and high-pass filters. The low-pass band usually contains the highest energy and the bands with one or more high-pass filters results usually contain comparatively less energy. Each of these bands can be decimated (by dropping, every other sample in the horizontal and vertical directions) without any loss of information.

Compression can be obtained by selective quantization of the four bands. Since the lowpass band contains the most energy, this band is quantized the least. Since the other bands contain less energy, they can be more heavily quantized without affecting the visual quality of the reconstructed image data. The low-pass band can be further compressed with a lossless technique, or can be recursively processed (prior to quantization) in the same way the original image was processed, thus reducing the low-pass band bit rate by a factor of four at every recursion level.

Figure 4.3.1.3-1 Subband/Wavelet compression of A to Ac (Image A Compressed)



In Figure 4.3.1.3-1, several stages of subband/wavelet decomposition are illustrated. The lowpass band, A,. is obtained by convoluting the input image, A, with the low pass filter, G, in both the row and column directions, followed by decimation (subsampling) by a factor of 2. Often a simple low-pass filter such as the Haar filter produces good results (Glover & Kwatra, 1992). After n stages of decomposition, the nth-stage low-pass band, A_n, is produced, plus several stages of high frequency detail images (D_{11}, D_{12}, D_{13}, ..., D_{j}l, D_{j2}, D_{j3}, ..., D_{n}i, D_{n2}, D_{n3}). The *j*th-stage high frequency detail information is contained in D_{j}l, D_{j2}, and D_{j3}, which result from convolving with the low-pass filter, G, and the high-pass filter, H, along rows and columns, as shown in Figure 4.3.1.3-1. The filtered result is decimated by a factor of 2. The outputs of the several decomposition stages are quantized and combined in a multiplexing scheme.

The reconstruction of the subband/wavelet encoded data is exactly the reverse process. The four channels are up sampled by a factor of 2, convolved with the appropriate inverse filters, and added. If the compressed image was the result of *n* stages of decomposition, *n* stages of synthesis are required to reconstruct the input image to the fullest detail.

Subband/wavelet encoding is often used when progressive compression/decompression is desired. Since A_n is a lower resolution version of the input image, A, it can be used in image browse applications. At an intermediate stage, *j*, A_j is an intermediate resolution version of the input image. The final result, A', is a full resolution reconstruction of the original image, with some loss in detail (as determined by the amount of quantization employed). However, this final result can be of visually lossless quality. Lossless reconstruction can be obtained if residuals (the differences

between the A and A') are losslessly encoded by some other method and combined with the final wavelet/subband synthesis result.

Combining subband or wavelet encoding with vector quantization (VQ) is a promising approach to compression of image data. Recently, Antonini *et al.* (1992) showed that a combination of wavelet encoding and VQ provided visually acceptable image reconstructions down to less that 0.5 bits/pixel. Even better results are reported (Barnes & Holder, 1993; Kossentini *et al.,* 1993) from combining multi-stage residual vector quantization (RVQ) with subband coding. An application of wavelet encoding to progressive transmission is given by Blanford (1993). Blanford showed that using wavelet encoding in his fine-grained progressive transmission scheme resulted in a faster decrease in the image's normalized MSE with percentage of data submitted when compared to two other encoding methods.

Most people making predictions as the future standards that will be produced by JPEG feel that some form of the subband/wavelet transform will be in the next standard. Most people researching in this field report the best results for visually lossless images and the lowest bit rates.

4.3.1.4 Multispectral KLT-Wavelet Data Compression

Epstein *et al.*, (1992) report on a hybrid approach for variable loss compression of multispectral image data using the Karhunen-Loève transform (KLT) and wavelet compression. They suggest that this approach is suitable for browsing multispectral image data. The approach is as follows:

i. Subtract the mean value from each spectral band.

ii. Compute the KLT from the n-band zero-mean image and apply it to the zero-mean multispectral image data.

iii. Apply the wavelet transform to each of the resulting spectrally decorrelated principal components.

iv. Quantize and code the spectrally and spatially decorrelated data using run-length encoding.

v. Apply Huffman coding to all the data.

The mean values and KLT eigenvectors were Huffman coded and prepended to the coded data. The amount of compression (and loss) is controlled by specifying the quantizer bin size in step iv.

Most multispectral data exhibits a significant amount of correlation between spectrally adjacent bands of data. This hybrid approach takes advantage of this spectral correlation through the KLT. The wavelet transform then exploits the spatial correlations between neighboring pixels.

This hybrid approach was tested on a 512-by-512 pixel section of a Landsat TM image of Kuwait. A compression ratio of nearly 11 was achieved for a quantizer bin size of 32, and a band average MSE of 6.85. This compares to a compression ratio of 7.5 and a band average MSE of 10.95 when the wavelet compression was applied directly to the image data (no KLT employed). A similar approach reported by Markas and Reif (1993) reported perceptually lossless image quality with a Landsat TM image of Washington, DC at compression ratios of 15 to 24 (depending on spectral band).

4.3.2 Vector Quantization

[Portions From Tilton *et al.,* 1993]

Vector Quantization (VQ) is the vector extension of scalar quantization, and is designed to reduce the bit rate over communications channels or to reduce data file volumes for storage (Gray, 1984). The VQ vectors are obtained from two-dimensional image data by systematically extracting non-overlapping blocks from the image. Such vectors allow VQ to exploit the two-dimensional correlations in the image data. (This exploitation is similar to the exploitation achieved in adaptive and predictive coding.) If the image is multispectral, non-overlapping cubes may be used, allowing VQ to also exploit the spectrum correlations. The blocks (or cubes) are converted to vectors by performing a (band by band) raster scan of each block.

VQ builds up a dictionary of a limited number of representative vectors, called codevectors, and then codes the image by using the index value of the closest code vectors from this dictionary, called a codebook, in place of each image vector. The codebook is optimal, for a given number of codevectors, if the image reconstructed from the codevectors is as similar as it can be to the original image over all possible codebooks of that size. If the Euclidean distance is used as the criterion for choosing the closest codevector to the image vectors, the image constructed from the optimal codebook will be the most similar image that can be so constructed in terms of Mean Squared Error (MSE).

Each codevector is represented by an address containing log2*m* bits, where m is the number of codevectors in the codebook. Assume vectors of size k are drawn from the input image and matched with those in the codebook. Using the addresses of the matched codevectors to represent the input image vectors results in a decreased rate of (log2*m*)/*k* bits/pixel or compression ratio (CR) of (*k***b*)/log2*m*, where *b* is the pixel brightness resolution in bits/pixel in the original image. In all practical situations the codebook size, *m*, is much smaller than *n*, the number of vectors extracted from the input image.

The most important phase of VQ is the training process in which an optimal codebook (by some error criterion such as least MSE) is learned from the input samples. The most widely used training algorithm is the Linde-Buzo-Gray (LBG) algorithm (Linde *et al.,* 1980). In this algorithm, the initial state of the codebook is randomly selected from the input sample set. The codebook is then improved iteratively by mapping all the samples from the training set onto the codebook and replacing the current codevectors with the centroids of all vectors that mapped onto each codevector. Convergence occurs when a minimal change occurs in the codebook from one iteration to the next. See Kohonen (1990) for an alternative training approach based on the "Self Organizing Feature Map."

Both the training and coding phase of VQ require finding the codevector which is the closest match to a given vector. Computing this closest match requires computations proportional to the size of the codebook, making this step computationally expensive. Computational cost can be reduced by employing suboptimal approaches such as the tree structured approach (Gray, 1984), or the pruned tree structured approach (Chou *et al.,* 1989). An alternative approach to reducing the processing time for training and coding is to implement these phases on a SIMD (Single Instruction, Multiple Data stream) massively parallel computer (Manohar & Tilton, 1992). If the number of processors in the SIMD machine is greater than or equal to the number of codevectors, the computations required for matching codevectors are proportional to the number of bits contained in a codevector, rather than the number of codevectors in the codebook.

VQ techniques are particularly suitable for data archives and distribution across computer network applications due to asymmetrical coding and decoding efficiencies. The coding is computationally expensive, but is a one time effort, and can be done at archival center using a large capacity machine. The decoding part, however, is a table lookup process which can be done at the user end very efficiently, so users are not burdened with computational difficulties.

Progressive Vector Quantization (PVQ) is a variant of VQ designed to provide multiple levels of compression (Manohar & Tilton, 1992). (See Section 4.2.7 Progressive Transmission) In PVQ, an image can be decomposed to several levels as long as this decomposition is economical for lossless compression. The tradeoffs are compression ratio and computational requirements. As the number of decomposition levels increases, the overall effective lossless compression ratio increases asymptotically to a maximum level, after which it may decrease. These results are reported in more detail in Manohar and Tilton (1993).

There are many other variations of VQ reported. Recent examples of interest are: a mean removed variation of weighted universal vector quantization (Andrews *et al.,* 1993), and fast VQ algorithms based on k-dimensional trees and on a neighborhood search algorithm (Arya and Mount, 1993).

4.3.3 Other Lossy Compression Methods

Some of the methods discussed in section 4.2 Lossless Data Compression Algorithms of this paper can also be implemented in a lossy manner. Predictive and adaptive methods can become lossy by not completely encoding the error signal. Many of the lossless methods have some sort of binary tree or hashing table in the implementation, and these methods can become lossy by using a pruned tree approach. Also, simple outline pictures can be made by simple filtering techniques.

4.4 Hybrid Techniques

A lot of data compression implementations combine techniques to achieve better results. For example, you can apply lossless data compression technique to a lossy data compression product to achieve additional reduction, or apply the lossless technique to the difference between original image and 'lossy' image to reconstruct the original image (See Section 4.2.7 Progress Techniques.) Methods can be combined to exploit the similarities between the spectral band as described in section 4.2.3 Adaptive and Predictive Coding and 4.3.1.4 KLT-Wavelet.

4.5 Bit Error Tolerance

More research and investigation needs to be done to determine the tolerance of these algorithms to bit errors. Certainly most bit errors would go undetected in most of these algorithms, and the ability of the algorithm to continue decompressing compressed data in the result an eronenous value is found in the compressed file will be largely implementation dependent.

4.6 Ongoing Research by GSFC

4.6.1 Model-Based Vector Quantization

[Portions from M. Manohar et al., 1994]

M. Manohar is currently researching a model based vector quantization MVQ technique. MVQ will eliminate the need to develop and store codebooks (as described in section 4.2.7) by generating the codebook internally using error models at the coding and decoding ends from a few image specific parameters. Two different models have been explored for mean removed vectors from the source images. The first one is a Laplacian multivariate (LMV) model and the second one is based on the first order Markov random field (MRF) model. In LMV model, the source statistics are obtained by computing the covariance matrix of the mean removed vectors. In MRF each vector element is the weighted sum of its spatial neighborhood elements and the a Laplacian random error. For the given source the optimal random field model parameters are learned by minimizing the mean squared error (MSE) between input image and reconstructed image.

MVQ has been tested on AVHRR data, and the results using both models are better than the results of VQ using codebooks generated by a training set which does not include the images to be coded. The Lossy JPEG standard produces a higher quality browse result as does VQ when the training set includes typical encoded images. The advantage of MVQ and VQ over JPEG is they are easy to develop for images that do not fit into the current JPEG standard (8 or 12 bits per pixel.)

4.6.2 Triada's, N-Gram product

GSFC is also currently trying to test a commercial product still in development by Triada corporation called the N-Gram. The N-Gram works as follows: A stream is parsed into sets of words according to rules that are empirically determined to be appropriate for the data type. The processor receiving the input word pattern searches its local memory to determine if the input word pattern was previously recorded. If it has previously occurred, a counter is incremented and a signal representative of the storage location of the pattern is sent to the subsequent processing level. If the pattern has not previously been recorded, it is assigned a place in storage, a signal representative of its new location is sent to the subsequent processing level, and a counter is increment to the value 1. The signals sent to the next processing stage are handled in a similar manner. That is, the signal goes through the same algorithm using different memory storage patterns recursively nine times.

The memory storage patterns become very large as the system is still learning the input stream. In theory (unless the input stream is completely random), the memory structure will eventually converge and need to grow no larger. At this point, Triada claims satellite image data can be losslessly compressed at a ratio of 73:1. This high compression is achieved by exploiting the similarities between the images.

GSFC has been trying to replicate Triada's claim and has been unable to do so because they have not had access to a large enough computer system. Triada claims it needed 3.6 GB of memory to compress 8 bit Landsat and NOAA data. There is a very real question of when and where does the memory structure converge. Also the 73:1 compression ratio claim does not include the size of the data kept in the memory structure. GSFC is still trying to verify Triada's claim.

Currently N-Gram is not commercially available and has no major corporate sponsor. Without a corporate sponsor it is doubtful if N-Gram will ever be a marketing success. The N-Gram product requires a very big super computer with a large amount of on board memory which makes it a very expensive compression technique to implement. N-Gram could only be cost effective for very large storage requirements and for data sets of a limited entropy level.

5 Applicability to ECS and Research Recommendations

5.1 How will ECS use data

Adopting a data compression methodology for ECS must be sensitive of ECS's needs. The ultimate data compression methodology will not necessarily be the method that yields the top compression ratio. More critical issues probably facing ECS are the processing demand of compression/decompression and the conveyance of decompression.

5.1.1 Other ECS elements

If compression is to be used in ECS beyond the archive (data server), the issue of how and where data will be decompressed becomes very relevant. If compressed data is sent to processing elements or other DAAC sites, these elements must decompress the data before processing can begin. As granule sizes become larger, this can become a very laborious task. A trade study will be needed to determine if the cost of additional processing power justifies the reduction in communications bandwidth. Another possibility to reduce network bandwidth, is to use network compression between ECS elements.

5.1.2 Client Usage

An even more difficult question to answer is, how clients will use and receive ECS data? Details of this question are supposed to be answered by the User Data Model. In the future, communications bandwidth will probably remain as expensive or change moderately in price compared to today; however, processing power will probably be much cheaper. Also if the user receives media distribution (also decreasing in price) of data, will the reduction in media and shipping cost offset the burden of the client to decompress the data? Ultimately some user will probably want to receive compressed data, and some users will want to receive uncompressed data.

One possible solution to reducing the complexity of dealing with compressed data to the client is to include the compression/decompression software in the IMS toolbox.

5.1.3 Hierarchical Data Format

Hierarchical Data Format (HDF) is also an issue. In HDF, the data is supposed to be layered so it is easy to reach into to file and pull out only the necessary data. If an entire HDF granule is compressed as one file, the entire granule would need to be decompressed to access the necessary data. Obviously a preferable approach would be to compress subsection of the file as separate entities so they could be decompressed as needed. An important question about HDF that needs to be addressed is where are HDF files made? Are HDF files built up on the from pieces at time of retrieval? In this case, it would be easy to keep parts of the file in a compressed format using hardware compression. Or, are entire HDF files stored in the archive? In this case if sections of the file are compressed separately, the complexity of compression and decompression routines increases particularly if hardware compression is being used

5.1.4 Browse Data

The browse data portion of the archive is perhaps one of the few areas where significant storage and retrieval costs saving can be realized because lossy compression is probably acceptable for most browse data sets. A subsampled 8 bits per pixel image can be reduced easily by a factor of 10:1 and still be visually lossless. However, it can be very challenging to determine an acceptable quality for browse images of each data set.

The HDF formats of browse data are also important to data compression for the reasons discussed above in section 5.1.3.

Some engineers have suggested that progressive browse techniques should be used to enable the client to request better and better quality images. Doing this could be expensive for storage. It is not known if users want this option or would prefer just to have visually lossless browse images. The difference in transmission time needs to be considered to determine if there is a significant advantage in sending low quality browse images with an option to send more data in the future versus just sending the visually lossless browse images. Another issue is whether or not scientists even want the capability of increasing the browse quality.

Progressive compression techniques for browse images can be taken to the extreme of receiving lossless data as described in section 4.2.7 by storing residual images. This can be very expensive from a storage point of view because of the added storage and retrieval costs of all of the residuals. Despite that the no redundant data needs to be stored actually resulting in more highly compressed data, it would be difficult to recommend that lossless data be built from several granules each time requested. Considering bulk of ECS's data needs will be for processing which is always required in a lossless format, lossless granules should only require a simple decompression routine of one granule at the most. For this reason, progressive browse techniques resulting in full resolution order will probably require the residual to be built as needed.

Another browse question is: will browse data be stored in an archive, or will it be built as needed? Certainly some browse data of more popular data sets will be stored in a browse storage area; however, there will likely be a cut off in temporal resolution or of data sets as to how much browse data will be readily available. Other browse images will probably need to be built from the lossless granule. Obviously the more compression employed in the browse products the more browse products can be stored by the ECS system, assuming browse products are kept in a separate area of the archive as currently envisioned.

5.1.5 Quick Look Data

What are the quality factors associated with Quick Look Data? For now, it will be assumed that quick look data will be produced using the same algorithm as browse products.

5.1.6 Issues Summary and Other Issues

Does the CPU burden to decompress justify the transmission or media savings?

What is the threshold of acceptable CPU burden to compression ratio?

What impact will bit errors within the archive have on a compressed file?

Is it better send compression files or use network compression to compress the packets?

Does ECS have a multi-resolution browse requirement?

If networked attached storage is implemented, will data be subsetted directly from the tape drives? Will compression of data prevent this?

5.2 ECS data and Compression

5.2.1 The Data Levels

5.2.1.1 Level 0 Raw Data

Depending on whether or not other sources (EDOS) archive all level 0 data, level 0 data needs to be treated differently then all other data. If ECS is the designated as the sole source archive of level 0 data, then the only compression option would be no compression. Any potential loss of level 0 from compression complications would not justify the cost savings of compressing level 0 data. Level 0 data is very compact any very hard to compress; therefore, it is possible that no compression is justifiable on level 0 regardless of which system is responsible for archiving it.

5.2.1.2 Level 1 Data

Level 1A and 1B data will largely be multidimensional arrays. Level 1 data is fairly compact by definition and in general will be hard to compress. Some data sets will probably be much more compressible then others depending on the entropy rate of the data the instrument collects. Level 1B data represents the greatest volume of the archived data. Level 1B tends to be image data which mean that the Rice algorithm (or other arithmetic coder with a predictor) should compress Level 1B data better then the Lempel-Ziv algorithm.

5.2.1.3 Level 2 Data

Level 2 data is derived from level 1 data and other level 2 data. Significant portions of level 2 data are raster and array data. Level 2 data contains more formatting information and is less compact then level 1 data; and therefore, level 2 data should be more compressible then level 1 data. However, each data set will probably perform differently, and dataset compressibility may vary significantly from granule to granule as the instrument detects different features of the planet. Level 2 data represents the second largest class of data.

5.2.1.4 Level 3 Data

Level 3 data is global gridded data so level 3 products of the same data set will have similar compression ratios. As with level 1 and 2 data, level 3 data compressibility will vary from data set to data set. There are very few level 3 products, and the combined total of the level 3 products is only a small portion of the archive.

5.2.1.5 Level 4 Data

Very little can be said about level 4 products trends. Level 4 products are modeling results and will vary greatly in diversity. Therefore, no conclusion about level 4 compressibility can be made.

5.2.1.6 Browse and Quick Look Data

Browse and quick look data products are all subsampled products from the above levels. As discussed in section 5.1.4 and 5.1.5, these data sets can be compressed using a lossy method. Currently there is no compression method which is significantly better then JPEG which is being used in V0 with compression ratios of 10:1. There are methods which will generally produce a better quality compression ratio then the JPEG lossy standard such as subband/wavelet encoding. A tradeoff study would be required to decide if it is justifiable to use a method other then JPEG. Given the conveyance and availability of JPEG, it is not very likely that another method should be employed. The fundamental limitation for JPEG is that it is only defined for 8 bits and 12 bits/pixel. If ECS has other browse requirements, other methods will have to be developed.

It is not trivial to come up with generic lossy compression algorithms as exists for lossless techniques. As new data sets develop, the quality factors associated with a lossy compression algorithm need to be established. If several spectral combinations are need, each combination may need to be calibrated separately because different kinds of features need to be preserved. All of this can become a nightmare from a quality assurance point of view. More information about browse and quick look requirements are needed to make a conclusive recommendation.

5.2.2 The Data Sets

The following data sets groupings are similar in nature and should have similar compression statistics.

AVHRR, Landsat, ASTER, MISR, and MODIS (Multispectral Radiometers) -- MODIS, ASTER and MISR are the largest sets (in that order) with sizes of 951GB/day, 455GB/day, and 263GB/day (not including reprocessing) respectively. MODIS is on two platforms AM-1 and PM-1 so the actual data rate when both are flying is 1902 GB/day. Landsat 7 is a non-EOS platform but will store about 122GB/day of data on ECS. All are multispectral data sets with level 1B radiance data being the largest portion. Interesting level 1B multispectral data from past NASA missions of Landsat and AVHRR can typically be compressed by UNIX compress by a factor of 1.5:1 to 1.8:1, and by using a successful predictor, multispectral images can be compressed from 2.5:1 to 2.8:1 or about a 50 to 60% improvement. The data model states that a MODIS level 1B granule will be 1.3GB, and MISR will have a level 1B granule sizes between 3 and 5GB. There are no compression statistics dealing with such large granule sizes, and it can not be said with authority that past compression statistics of multispectral images are valid for granules of such large sizes. Such large sizes may not compress well using hardware compression schemes that use a limited dictionary size such as the modified LZ2 algorithm. Therefore if hardware compression was desired to compress these large granules, arithmetic coders such as IRDC and the Rice algorithm would probably be preferable. From a compression/decompression point of view, clearly these granules should be broken up into smaller pieces before storing on the ECS system if there is a significant demand to have these granules subsetted or partitioned by clients. If these granules are stored as whole pieces, the entire granule will have to be retrieved and decompressed to access any subset of the granule.

The ASTER, MISR, and MODIS data sets represent almost 95% of the EOS data. If these data sets can be compressed successfully all other compression discussion is somewhat moot because of how little data the remaining data sets represent. Almost 70% (not including reprocessing) of

these three data sets is level 1A and 1B data which is very difficult to compress because the data is so compact.

ERBE is similar to CERES. CERES is the 4th largest EOS data set at over 10GB/day on TRMM, over 21GB/day on PM-1, and over 21GB/day on AM-1. It is a small data set when compared to MODIS, MISR, or ASTER.

AIRS, AMSU, and MHS are multi-channel sounders of different frequency ranges. AIRS is the 5th largest EOS data set at 48GB/day. AMSU and MHS are small in comparison producing less then 0.2GB/day each.

GLAS is the next largest EOS data set at just under 10GB/day.

SSM/I is similar to MIMR. MIMR produces just under 6GB/day of data.

Nimbus-7 and CZCS are similar to SeaWiFS or EOS-Color. CZCS is a 6 band color scanner and has similar compression properties as the 7 multispectral band Landsat images (see above discussion). SeaWiFS II is a small data set at 2GB/day and will have typical compression ratios and compressibility of MODIS, MISR, and ASTER.

With the exception of HIRDLS which produces 1.3GB/day, the remaining EOS instruments produce less then 1GB/day each and are insignificant in size compared to the above data sets.

5.3 Recommendations for Prototyping and Research

5.3.1 Landsat and AVHRR Level 1B Study

AVHRR and Landsat TM Level 1B data could be studied to determine which lossless compression method exceeds in compressing it. These results may theoretically be the same as the future data sets of MODIS, MISR, and ASTER, and Landsat 7 level 1B data.

A sampling of AVHRR and Landsat TM full resolution granules from the surface of planet would be needed for the study. The Comparative Lossless Compression Package (CRUSH) developed by Hughes STX for Goddard could be used to compare LZW, WNC, Adaptive WNC, two windowed versions of LZ, Rice, and lossless JPEG. The CRUSH algorithm will provide processing rate and compression ratio for all the above mentioned lossless techniques.

The study would hopefully conclude whether or not it is justifiable to use a different algorithm then LZW (UNIX Compress) for level 1B data for the largest data sets.

5.3.2 Derived Products (Level 2 through Level 4)

Very few assumptions can be made about derived products compressibility. It will not be possible to ever conclude that one compression algorithm is always better then another algorithm. It may even be impossible to predict that one algorithm is consistently better then another algorithm by volume. It cannot even be assumed that derived products will consistently be image data or multidimensional integer data.

A cost analysis is necessary to determine if it is justifiable to use more then one compression method. That is does the possible compression savings justify the complexity of implementing more then one algorithm.

5.3.3 Browse and Quick Look Data

More information about browse and quick look requirements need to be gathered before a recommendation for future study of lossy techniques can be made.

5.3.4 Test COTS compression chips

Several COTS compression chips are available today. These chips could be acquired and tested for speed and effectiveness against the software algorithms they implement.

5.3.5 Other Research Ideas:

Ask DAACs using compression to log their compression statistics at time of compression and archival to get more meaning full results on average compression of the LZW (UNIX compress) algorithm. That is gather global coverage statistics.

Small study on the effectiveness of the DCLZ algorithm with a limited dictionary size as available on COTS chips today. Compare these results with the above log of the LZ1 algorithm.

Explore band ordering and adaptive techniques to exploit the similarities of the spectral bands that is common in many ECS data sets. (See Tate, 1994)

Compare hardware versus software compression. How much CPU demand do these compression/decompression algorithms use? Can clients and the PGS tolerate the cost to decompress?

Continue exploring COTS compression chips applicability and effectiveness.

What effect does sending compressed data across the CSMS have? If the CSMS also uses compression will the CSMS automatically recognize compressed data and not attempt to compress any further?

Research the applicability of JPEG lossy compression for the production of browse products. Attempt to determine if it is justifiable to use a different lossy algorithm for the production of browse products.

6. Early Conclusions

LZW (UNIX Compress) will probably average 2:1 compression for all EOS data maybe less. Despite complaints that UNIX compress does not yield a high compression ratio compared to other algorithms, a GSFC concluded that there was no single algorithm that always worked better on a granule by granule basis. However when considering an entire data set instead of individual granules some algorithms may perform better. If multiple algorithms are employed, an average data compression ratio of 3:1 may be obtainable for EOS data.

Appendix A:
Abbreviations and Acronyms

ECS	EOSDIS Core System
EDOS	EOS Data and Operations System
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
DAAC	Distributed Active Archive Center
DADS	Data Archive and Distribution System (ECS)
DCLZ	Data Compression Lempel-Ziv algorithm (Dictionary Based)
DCT	Discrete Cosine Transform
DPCM	Differential pulse code modulation
FDCT	Forward DCT
HDF	Hierarchical Data Format
IDCT	Inverse DCT
KLT	Karhunen-Loève Transform
JPEG	Joint Photographics Experts Group
LMV	Laplacian Multivariate
LZ	Lempel-Ziv
LZ1	Lempel-Ziv 1977 algorithm
LZ2	Lempel-Ziv 1988 algorithm
LZW	Lempel-Ziv-Welch
MRF	Markov random field
MVQ	Model Based Vector Quantization
NCSA	National Center for Supercomputing Applications
RLE	Run Length Encoding
SAR	Synthetic Aperture Radar
SNR	Signal to Noise Ratio
V0	Version 0 of ECS
VQ	Vector Quantization
WNC	Witten-Neal-Clearly

Appendix B: References

Antonini, M., Barlaud, M., Mathieu, P, and Daubechies, I. 1992. Image coding using wavelet transform. *IEEE Trans. on Image Processing.* 1(2):205-220.

J. Tilton, M. Manohar, and J. Newcorner, "Earth Science Data Compression Issues and Activities," GSFC.

Chiang T., "Data Compression White Paper", Loral Aerosys.

Bugajski J., "A Petabyte Size Electronic Library Using the N-Gram Memory Engine, "IEEE/NASA Mass Storage Conference, October 1993.

M. Manohar, J. Tilton, B. Kobler, and P. Hariharan, "Model-Based Vector Quantization," GSFC.

AHA Application Note, "Primer: Data Compression Lempel-Ziv (DCLV)," Moscow, Idaho.

S. Tate, "Band Ordering in Lossless Compression of Multispectral Images", *Proceedings Data Compression Conference* (IEEE Catalog Number 93TH0626-2), March 29-31, 1994, pp. 311-320.

Chou, P. A., Lookagaugh, T. and Gray, R. M. 1989. Optimal pruning with application to tree structured source coding and modeling. *IEEE Trans. on Info. Theory.* 35(2):299 315.

Gray, R. M. 1984. Vector Quantization. *IEEE ASSP Magazine.* pp. 4 28, April 1984.

Howard, P. G. and Vitter, J. S. 1993. Fast and efficient lossless image compression. *In Proc. Data Compression Conference* (Catalog Number 93TH0536-3). March 30-April 2, 1993, Snowbird, Utah, pp. 351-360.

Huffman, D. A. 1952. A method for the construction of minimum redundancy codes. Proc. IRE. 40:1098:1101.

Kohonen, T. 1990. The self-organizing map. *Proc. of the IEEE.* 78(9):1464-1480.

Manohar, M. and Tilton, J.C. 1992 Progressive vector quantization of multispectral image data using a massively parallel SIMD machine. *In Proc. Data Compression Conference* (Catalog Number 92TH0436-6). March 24-27, 1992, Snowbird, Utah, pp. 181-190.

Manohar, M. and Tilton, J. C. 1993. Progressive vector quantization on a massively parallel SIMD machine with application to multispectral image data. Submitted to *IEEE Trans. on Image Processing.*

Markas, T. and Reif, J. 1993. Multispectral image compression algorithms. *In Proc. Data Compression Conference* (IEEE Catalog Number 93TH0536-3). March 30-April 2, 1993, Snowbird, Utah, pp. 391-400.

Pennebaker and Mitchell. 1993. JPEG Still Image Data Compression Standard. New York: Van Nostrand Reinholt.

Rice, R. F. 1979. Practical universal Noiseless coding. *SPIE Symposium Proceedings.* Vol. 207, San Diego, CA, August 1979.

Rice, R. F., Yeh, P. S. and Miller, W. H. 1991. Algorithms for a very high speed universal noiseless coding module. *JPL Publication* 91-1. JPL, Pasadena, California, Feb. 15, 1991.

Wallace, G. K. 1991. The JPEG still picture compression standard. *Communications for the ACM*. 34(4):31-91.

Welch, T. A. 1984. A technique for high-performance data compression. IEEE Computer. 17(6):8-19.

Witten, I., Neal, R., and Cleary, J. 1987. Arithmetic coding for data compression. *Communications for the ACM*. 30(6):520-540.